AFRL-IF-RS-TR-2006-313
Final Technical Report
October 2006

USING HETEROGENEOUS HIGH PERFORMANCE COMPUTING CLUSTER FOR SUPPORTING FINE-GRAINED PARALLEL APPLICATIONS

SUNY Binghamton

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO FINAL REPORT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Rome Research Site Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-IF-RS-TR-2006-313 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/   /s/

STANLEY LIS   JAMES A. COLLINS, Deputy Chief
Work Unit Manager   Advanced Computing Division
                                             Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
OCT 2006	Final	Feb 05 – Jan 06

4. TITLE AND SUBTITLE

USING HETEROGENEOUS HIGH PERFORMANCE COMPUTING CLUSTER FOR SUPPORTING FINE-GRAINED PARALLEL APPLICATIONS

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA8750-05-1-0130

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Nael Abu-Gazaleh

5d. PROJECT NUMBER
NBGQ

5e. TASK NUMBER
10

5f. WORK UNIT NUMBER
09

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Dept.
SUNY Binghamton
Vestal Pkwy East
Binghamton NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFTC
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-313

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 06-729

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The use of a heterogeneous cluster comprised of host processors and Field Programmable Gate Arrays (FPGAs) was investigated for accelerating the performance of parallel fine-grained applications using a direct FPGA to FPGA communication channel. The communication channel is implemented with an all-to-all board that attaches directly to the FPGA boards via their I/O interface. Test scripts were written to test the all-to-all board. The necessary communication support was designed, tested, and implemented to allow message exchange over the all-to-all board. The all-to-all support provides a low latency, low-bandwidth, communication channel for the FPGAs that can considerably extend the range of parallel applications. The Parallel Discrete Event Simulation was used to demonstrate that this computing model can accelerate the performance of parallel applications.

15. SUBJECT TERMS

Heterogeneous cluster, FPGA, Parallel Discrete Event Simulation, all-to-all board

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Stanley Lis
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UL	23	19b. TELEPHONE NUMBER (*Include area code*)

Table of Contents

List of Figures

1 Preface

At AFRL, a heterogeneous cluster is available with Field Programmable Gate Array (FPGA) boards attached on the I/O board. For parallel applications, especially fine-grained ones, the performance of the application is often defined by the performance of the communication subsystem. The communication between the host processor and the FPGA on the same node requires an I/O operation that is expensive. Further, FPGA to FPGA communication across different nodes must go through the host processor. As a result, the use of this model for parallel applications is severely limited: the communication cost is prohibitive for most applications. In an attempt to alleviate this bottleneck, a previous effort at AFRL designed a direct FPGA to FPGA communication channel (all-to-all board) that attaches to the FPGA boards directly via their I/O interface. Pin and board limitations suggested a design which connects all the FPGA boards to each other via a dedicated serial line: each board has 2 lines to each other board, one for incoming and one for outgoing data. The all-to-all support provides a low latency, but low-bandwidth, communication channel for the FPGAs that can considerably extend the range of parallel applications that can benefit from this infrastructure.

2 Abstract/Summary

The goal of this project was to investigate this new infrastructure and develop support for basic communication through it that can be reused by other application developers. In addition, our goal was to demonstrate the capabilities of the new infrastructure using Parallel Discrete Event Simulation as an application. This work builds on an initial design the PI created during prior summer faculty research visits to AFRL Rome. The all-to-all board was being fabricated as the project started, and became available a few months into the project. The grant supported a Research Assistant (David Curren) effort for 20 hours a week for the duration of the grant, with the exception of 12 weeks in summer where he was supported as a summer graduate student working on the same project.

The first 3 months of the performance period were spent in training the student in various technologies and other background needed for the project. This includes: VHDL, the physical design process and the tools, the PDES simulator, the preliminary design for the communication unit and relevant parallel architecture support.

The first task was to develop a methodology and test scripts to test the all-to-all board. The board is very large physically and has a large number of wires. Thus, cross-talk is an issue that was anticipated and the board was designed to tolerate cross-talk up to 200MHz. However, in testing the board, we discovered that cross-talk arises for some patterns at data rates above 51MHz. Some sparse data rates work up to 80 MHz, but in the worst case, with continuous communication of randomly generated patterns across all wires, only 51MHz or lower could be sustained.

The second task was to take the preliminary design for the communication support and test it and debug it if necessary. The preliminary communication support was

developed without the all-to-all board, and therefore was only partially tested. The design supported variable size messages in chunks of 48-bits. In practice, this task proved to considerably more difficult than we anticipated for a number of reasons, including: (1) race conditions that arose due to asynchronous clocks (with the clock coming off the all-to-all board, and the clock on the FPGA. These were extremely time-consuming and difficult to isolate and resolve; (2) excessive critical path delays that arose primarily due to the large distances between the on FPGA buffer messages before they are dispatched to the I/O board or collected by the FPGA; (3) Delays in the scheduler used to arbitrate a bus used to pull messages from the FIFO buffers to the FPGA. Initially, the design was based on a round-robin scheduler with a propagating dependency (similar to that of carry propagate adders). We worked on a design for a pipelined version (similar to carry look-ahead adders), and eventually determined that a priority encoder design can do the same function; (4) Synchronization issues across the different boards. As the boards start, they will generally not be synchronized to the same 48-bit chunk boundary required by the design. We resolved this problem through a simple distributed protocol where each node signifies it is awake by transmitting a special bit pattern. When all nodes are awake, a designated node sends a pattern that resets the 48-bit clock on all nodes concurrently.

Finally, the major obstacle that severely hampered our ability with task two is the fact that the design tools were unusable across the network. All the tools had to be run on the AFRL machines where the licenses were. However, exporting X across the network was extremely slow to the point where the tools that had graphical interfaces could not be used at all. Thus, we were reduced to testing the design by extracting portions of the design, and running on the FPGA to analyze the behavior via the observed output—an extremely time-consuming and error-prone process.

The third proposed task was to demonstrate that the infrastructure can be used to accelerate Parallel Discrete Event Simulation. We debugged and improved an initial design for Global Virtual Time (GVT) computation that was developed by the PI over the 2004 and 2005 summers as a visiting researcher to AFRL/IF. We also debugged the interface with the I/O support developed in the second task, and with the simulator via the PCI bus. Full integration with the simulator was not accomplished due to the unexpected and time-consuming challenges in the second task.

3   Introduction and Overview

Fine grained and dynamic applications such as Parallel Discrete Event Simulation [Fujimoto 1990, Metron] present a challenge to parallel processing environments in general, and clusters in particular for the following reasons: (1) The fine grained nature makes the execution communication bound; since the cost of communication is significantly higher than computation, frequent communication limits the performance and scalability of fine-grained applications; and (2) Their dynamic nature means that effective partitioning and system configuration are difficult to develop. The simulation engine typically has several configurable parameters that can have a dramatic influence on performance, including parameters governing the degree of optimism, the

frequency of state saving, rollback implementation and others. Finding an appropriate configuration is difficult; moreover, a good initial configuration may not be effective as the simulation behavior changes. Similarly, it is difficult to achieve effective initial model partitioning configurations, and to evolve these as the simulation behavior changes. As a result, it is difficult to achieve large speedups in cluster environments

Computer Modeling and simulation have grown to become a powerful approach for complex system design and analysis. Parallel Discrete Event Simulation (PDES) is an approach to parallelizing simulation to increase its performance and capacity, allowing the simulation of bigger more detailed models and more interesting scenarios in a given time budget. PDES underlies several areas of interest for the DoD including war-gaming, planning and decision making, and complex system design and analysis, including both hardware and software systems.

In PDES, a simulation model is partitioned across several logical simulator processes (or LPs). Each LP processes its events in time-stamped order. Synchronization among different LPs may be achieved using one of two major approaches: (1) conservatively: an event at an LP is processed only if all other LPs guarantee that it can be processed safely (no events earlier than it will be generated to that LP); (2) Optimistically: LPs process events without concern for causality. Events received from other LPs with a time stamp earlier than the current simulation time signal a causality error. Such errors are recovered from by rolling back the local simulation state to a time earlier than the received straggler event, and sending out messages canceling any messages that were sent out erroneously. To be able to achieve this synchronization, each LP must periodically checkpoint its state and event information. Checkpoints are garbage collected when they are no longer needed (when the global simulation time has passed them). This requires computing the Global Virtual Time (GVT) of the simulation to determine which history information may be garbage collected. Fujimoto [Fujimoto 90] wrote an excellent survey on PDES and PDES optimization approaches.

In previous efforts to accelerate the performance of PDES [AbuGhazaleh04], we identified that the communication subsystem is a major bottleneck in PDES performance. In addition, initial efforts in exploiting the FPGAs on a Heterogeneous High Performance Cluster (HHPC) to accelerate the performance of a PDES simulation were reported. Using FPGA boards to accelerate the performance of some critical simulation subsystems was the goal of the study. Since PDES is a fine grained operation, and the communication with the FPGA board expensive, it is almost impossible to use the FPGAs to optimize the simulation kernel, which requires message exchanges potentially with every event. We conjecture that other parallel applications would face the same problem in trying to use this infrastructure.

In response to this limitation, we needed to create an alternative channel for the FPGAs to communicate without having to interrupt the primary host processor. To achieve this, a serial all-to-all connector board that provides direct, low bandwidth, low latency, connectivity among the FPGA boards, was designed. This board provided a channel for the FPGAs to communicate directly, potentially greatly improving the performance of fine-grained applications with components of the computation residing

on the FPGAs. To demonstrate such an application, we identified the Global Virtual Time computation as a target for FPGA implementation. We use an algorithm for GVT computation based on Mattern's two-phase algorithm. Each node provides local time and message counts when it enters GVT computation phase and whenever transit message count changes to the FPGA board. The boards communicate among each other to detect the global messages in transit count. When that reaches 0, they compute the minimum of the local times and broadcast it to all the host processors. We considered several other ideas using the same infrastrucutre. For example, the low-latency direct connectivity between the FPGAs is ideally suited for monitoring/exchange of control messages. Such capability will allow us to carry out adaptive control of the simulation configuration.

Before such implementations can be undertaken, the all-to-all board must be tested for functionality and performance to set the baseline physical rate it can communicate on. Further, support for communication using the all-to-all board must be developed; the equivalent for the link layer for this communication channel. To this end, the goals of this project were:

1. Functionality and performance testing of the all-to-all board. The board is physically large, and contains a large number of wires; crosstalk is a major concern. The design accounted for cross-talk to be able to run, theoretically, at 200MHz. The testing should verify the actual connectivity as well as the effective maximum transmission rate before cross-talk arises for a number of communication patterns.
2. Design of the communication support to allow message exchange over the all-to-all board. The all-to-all board essentially is a physical medium that provides serial all-to-all connectivity across the FPGA boards. To allow message exchange, support must be built for exchanging messages over the medium. This includes defining the API and message formats for the communication, and implementing support for allowing the exchange of messages. In conventional networks, this includes: (1) Framing: defining where messages begin and end; (2) Reliability: was not considered; (3) encoding; and (4) medium access. It also includes buffering messages and exchange of messages with the application.
3. Integration with the PDES simulator: this includes verifying the GVT implementation on the FPGA.

We were able to accomplish the first goal. In addition, we developed a nearly functional communication support. We greatly underestimated the complexity of the design for goal 2. Primarily, the presence of multiple clocks (PCI bus clock, internal FPGA clock and I/O clock) controlling logic at the boundary between these three components, lead to very difficult to resolve race conditions. Other difficulties arose that were not caught by the simulator. Finally, tool issues greatly complicated the development and debugging efforts. As a result, the second goal was accomplished, with minor issues remaining to be resolved, but the third goal was not completed.

The remainder of this report is organized as follows. Section 4 presents some background material related to PDES, and HHPC. Section 5 describes the testing activity. Section 6 overviews the design. Section 7 describes SPEEDES and GVT

calculation.   Finally, Section 8 presents some concluding remarks.

4   Background

In this section we first overview PDES, and motivate the need for more effective fine-grained communication and self-monitoring and adaptation.   We then describe the HHPC environment in more detail.

4.1   Parallel Discrete Event Simulation

In Discrete Event Simulation (DES), a model starts with an initial state and an initial number of scheduled events. Events are ordered by simulation time in an event queue. Simulation proceeds by processing the earliest event, which can cause changes in the simulation state and schedule one or more future events. The simulation time advances to the time of the earliest unprocessed event. Simulation terminates when there are no more events to process or when a predetermined simulation time is reached.

 Parallel Discrete Event Simulation (PDES) leverages parallel processing to attempt to accelerate the performance and capacity of DES. The simulation model is partitioned across multiple simulation processes (called Logical Processes, or LPs).   Each LP maintains a local event queue and carries out simulation as described above (repeatedly processing the earliest time stamp event). A locally processed event may generate events to remote LPs (that is, affecting changes to state managed by the remote LP).   Thus, LPs communicate by exchanging time-stamped event messages [Jefferson-85]. Correct simulation requires that all events are processed in time-stamp order. Therefore, a synchronization model is needed to ensure that remote events are processed in time-stamp order.

Conservative PDES simulators carry out synchronization as follows: an LP, LP_i, does not process an event until it is guaranteed that no other LP will generate a remote event destined to LP_i with a time stamp earlier than the current earliest event.   Thus, explicit time step synchronization of the LPs is needed, severely limiting the potential event processing concurrency. Alternatively, optimistic PDES simulation (the so-called Time-Warp model [Fujimoto-90]) does not require explicit synchronization among LPs. Each LP enforces causal ordering on local events (by processing them in time-stamp order). Causality is preserved on remote events by detecting causality errors (when a *straggler* event with a time-stamp in the past is received) and recovering from them by rolling-back the simulation to a state prior to the time of the straggler event.   Thus, each simulator must maintain state and event histories in order to enable recovery from straggler events. This state information must be garbage collected to control memory usage and enhance the simulator locality; a Global Virtual Time (GVT) algorithm is used to detect the global progress time of the simulation, allowing the garbage collection of histories earlier than this time.

We use the SPEEDES simulation environment [Metron] in our studies; SPEEDES is a state of the art PDES simulator that uses Breathing Time Warp (an optimistic simulator that bounds optimist to attempt to control excessive rollbacks) [Steinman 1993]. More specifically, in order to prevent uncontrolled and erroneous optimism and its resultant

local loss of efficiency (and the pollution of the simulation at remote nodes via wrong events) each LP stops simulation after a configurable number of events Nopt is reached. Furthermore, before the Nopt limit is reached, another configurable (lower) limit Nrisk determines how many events are processed but with sending remote events. Thus, the first Nrisk events after GVT are processed optimistically, then the remaining Nopt-Ngvt are processed optimistically but without sending remote messages. Events after Nopt are not processed. Another configurable constant Ngvt determines how often GVT computation is initiated. SPEEDES is used in several projects in the Air Force and DoD.

The performance of PDES is heavily influenced by the message exchange latency: PDES is fine-grained, with event messages generated frequently (depending on the model) to remote LPs. Delays in receiving these messages can cause the simulation to be erroneous at remote LPs (since the event message will be received after the simulator has moved past it).   It has been shown that improving the performance of the communication subsystem results in significant improvement in the simulation performance (e.g., [Chetlur98,Sharma99,AbuGhazaleh04]). Furthermore, slow message exchange causes GVT estimates to be slow, limiting the available concurrency in Breathing Time Warp and reducing the efficiency of garbage collection (increasing the simulator memory footprint and worsening the average memory access time).

In addition, the simulation behavior is dynamic and unpredictable. There are several parameters that configure the simulation and the sub-algorithms used in it (for example, several parameters are used in Breathing Time Warp to control the degree of tolerated optimism); a suitable configuration depends on the current model behavior and can have a large effect on the simulation performance. Moreover, the model partitioning affects the resulting remote dependencies and can also significantly influence performance. Furthermore, the dependencies in the model can evolve dynamically (for example, as objects move away from initial close objects and closer to other objects). Thus, effective configuration and partitioning is necessary both initially and dynamically during run time. This polymorphic capability of the simulator is a long term goal of our research.   There is some prior work in adaptive control of PDES simulators [Radhakrishnan98] and on dynamic partitioning of the simulation [Avril96], some have been model-specific and none have been in the context of a Breathing Time Warp model or utilized the specialized hardware that is available to us. Moreover, we know of no prior work on morphing the underlying communication subsystem to match application behavior.

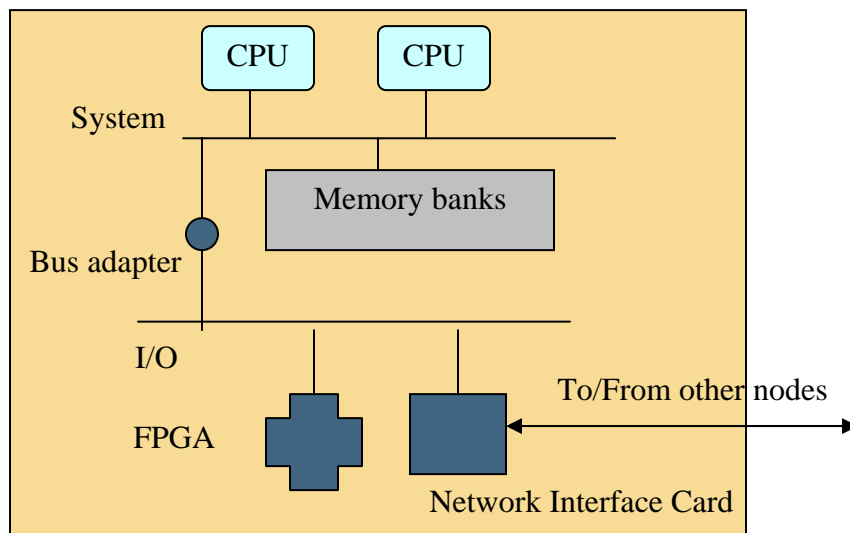## 4.2   Heterogeneous High Performance Computing (HHPC) Cluster

Figure 1: Architecture of the individual Node

HHPC is a Beowulf cluster made of COTS PCs (featuring dual processor Xeon's) interconnected via a Gigabit Ethernet Network and a Myrinet network [Boden 1995]. In addition, each node has an Annapolis Micro Devices (AMD) Wildstar II FPGA board on the PCI bus. The Wildstar has a Xilinx Virtex II FPGA, some DRAM and SRAM banks and an LVDS I/O card. We use the I/O card to interconnect the FPGAs directly to each other using a custom built all-to-all serial board. This board provides connectivity from every node to every other node concurrently using a dedicated serial line. This results in a low-latency but low-bandwidth communication channel among the FPGAs. Without this connectivity, all communication must go through the communication fabric at a latency ranging of around 10 microseconds (for the expensive Myrinet) to several 10s of microseconds for the commodity Gigabit Ethernet. We note that for this project we do not exploit the FPGA component of the cluster.

Typically, FPGA boards are used to accelerate sequential or high-granularity parallel applications that have high data parallelism or unusual data-paths (e.g., Image processing or Cryptographic applications). PDES does not fit this profile: it is fine-grained and does not, in general, require high data parallelism.

The FPGA boards reside on the I/O bus for the cluster and are not able to master this bus (see Figure 2). This forces all FPGA communication to go through the primary host, making interaction with the FPGA or across FPGA boards expensive and limiting the utility of the FPGAs for fine-grained applications. To remove this limitation we designed and all-to-all serial communication board that connects the FPGA boards on the different cluster nodes to each other directly through the I/O connector for the

7

boards. This connectivity provides direct conduit for communication between the FPGAs that does not require host interference. While this channel allows low latency communication, it is bit-serial and unsuitable for exchanging large messages.

5 All-to-All Board Testing

Because the all-to-all board had not been used before, it first needed to be tested. The all-to-all board is designed to be used with up to 24 computers, where one of the computers is designated to act as a clock for the rest of the nodes. Because there are a large number of connections handling fast signals, cross talk between the wires is a potential problem. The designers hoped to make the all-to-all board reliable up to approximately 200 MHz.

5.1 Preliminaries

There were some problems to overcome before even beginning testing. First, because the all-to-all board had not been used before, and no one at the site had experience with it, we had to first figure out how to communicate a signal over the board. Each FPGA card connects to the all-to-all board through an LVDS card. The software suite used by the FPGA card was supposed to include templates for communicating with the LVDS card; however, we did not have a copy of the software. At first, we were not aware of this and tried to write a template that would work with the LVDS card, but without the hardware schematics, we did not have much success. Meanwhile, we contacted Annapolis to obtain a copy of the template to use the LVDS card. We did receive the template before we were able to write our own version, and decided to use that. The provided template turned out to work well with our LVDS card, and we were able to detect that there were some signals crossing the board.

The next problem was figuring out how to control the signals being sent. At first, we only had four nodes connected to the all-to-all board. Because one of these nodes had to act as a clock for the other nodes, we had one node send its clock signal over the all-to-all board to every other node. We had the three other nodes available listening, one all-to-all input at a time, for this clock pulse. Eventually, we were able to locate the clock signal on all three nodes. Next, using this clock signal, we tried to send very small messages from one node to another. We did not know at the time which node was connected to which slot on the all-to-all board, so this, again, took a good amount of guesswork, and testing. We were eventually able to control the signals being sent and received. Infrequently, there were random numbers that would be received unpredictably. After some testing, we discovered that any time a node deprograms the FPGA card it is using, undefined signals are received. This was only a problem if nodes were deprogramming their FPGA cards while tests were still running and could easily be worked around.

5.2 Pin Mapping

The final major problem in getting the all-to-all board working was to connect all nodes to the all-to-all board, and figure out which LVDS connection corresponded with sending

and receiving port for each of the nodes. While it was time-consuming setting up all of the connections and checking that they worked, the process turned out to be fairly easy. Each connection to the all-to-all board is numbered from 0 to 22, with a 24th slot denoted as the clock. Node x sent to the node at all-to-all board position 0 through LVDS connection 0, position 1 through LVDS connection 2, and position n through LVDS connection 2n. Node x listened to the node at all-to-all board position 0 through LVDS connection 48, position 1 through LVDS connection 50, and position n through LVDS connection 48+2n. The reason that the LVDS connection numbers are all even is because the LVDS card uses the differential between two of its pins to send a message, meaning that two of its input pins are designated for use with every one of its output pins. Because a node does not send to itself over the all-to-all board, if the all-to-all board position n is greater that x, then you have to subtract two from the LVDS connection number. For example, if the node at all-to-all slot 0 wants to send to the node at position 1, it sends through LVDS pin 0.

5.3 Testing

With the all-to-all board successfully sending signals, we were able to start performing workload testing. A number of issues had to be considered when testing the all-to-all board. Each node had to be sending a different bit sequence to every other node. Not only would node 1 be sending a different pattern than node 2, but also the bit pattern being sent by node 1 to node 2 had to be different than the bit pattern being sent from node 1 to node 3. If two wires that would otherwise be experiencing cross talk are sending the same bit, both remain unchanged. Additionally, rather than having each node handle 23 incoming streams at once, which would have involved a complicated receiving mechanism, every test run involved all nodes listening to one predetermined connection. The tests were cycled so that every node eventually listened to every other node. In order to reduce the threat of cross talk, all nodes sent their bit pattern to all other nodes, even if the receiving node was not expected to be listening. Testing a number of different randomly generated bit patterns left little room for doubt of the results. This set of testing was called the worst scenario tests and were the most interesting performed. Other tests established that the all-to-all board did work for the, and tested low-traffic test conditions.

Some early, low traffic tests did in fact work up to speeds of near 200 MHz, some even exceeding this speed. However, the worst-case scenario tests were not as successful. In general, tests were successful up to 72 MHz, however, for 100% reliability, which is necessary for the simulator, the clock had to be slowed to 51 MHz. A second test was performed where node n sent the same message to every other node. This test was more reflective of what would happen in the GVT calculator for SPEEDES (although not necessarily for any application to be run over the all-to-all board). The results were similar. Most tests succeeded up until 81 MHz, but again, to provide 100% reliability, the clock needed to be set to 51 MHz or slower.

It is unclear why the design goal of 200MHz operation without crosstalk was not achieved. The testing scripts that were configurable in terms of parameters such as duration and test patterns were completed and delivered. The tests also compare the observed output against the sent messages and detect errors in message transmission.

5.4 Bootstrap and Synchronization

The design of the communication support (or I/O unit) requires that all the nodes should be all synchronized on 48-bit boundary. Since the nodes start asynchronously, the synchronization is not present by default. We developed a simple handshake algorithm where all nodes that wake up repeatedly announce their presence to a designated master node. Once all the nodes participating in the current application are awake, a synch start signal is broadcast to all to achieve synchronization. This implementation was designed and tested.

6   Overall Design and I/O unit activities

Exchange messages with other FPGA boards through I/O connector and all to all card

Message count and time updates from other nodes; Send our updates to them

I/O unit

Data Path and Internal Registers + GVT computation unit

Interface to host to receive Updates/requests and reply Status/GVT value

I/O Unit:
- Dual ported FIFOs, two for each other node (in/out)
- Serial ⇔ parallel conversion for data to/from I/O connector
- State machines to control data exchange to data path and I/O connector

GVT Unit:
- Reduction tree to add message counts from all nodes
- Reduction tree for computing minimum time once message count reaches zero
- State Machines to exchange updates and requests with datapath
- State Machines to control "color" change

Data Path
- State Machines to control communication with host through PCI bus
- Generate update messages to other hosts through I/O unit
- Receive and parse messages from the I/O unit; pass updates to GVT unit
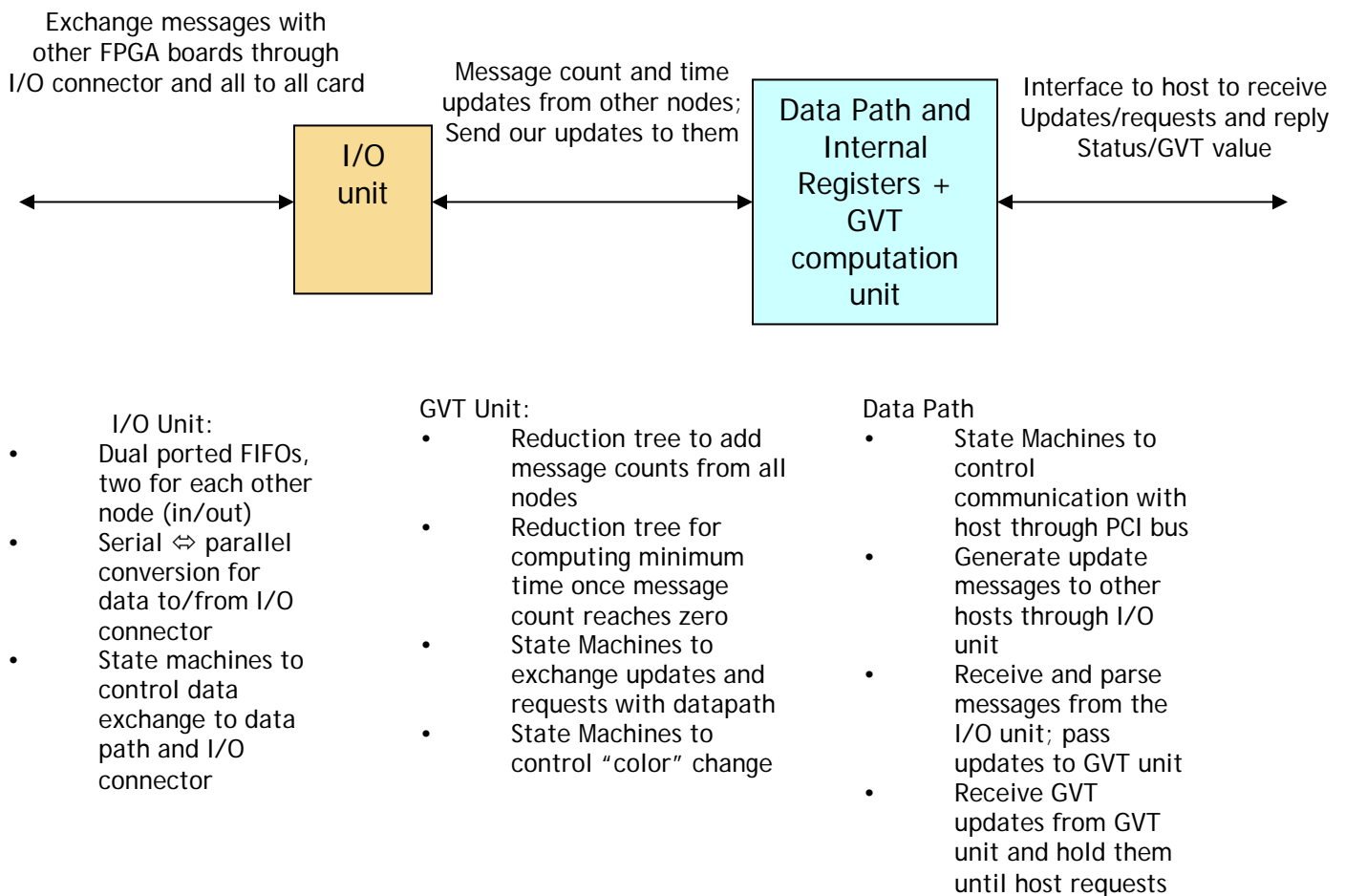- Receive GVT updates from GVT unit and hold them until host requests

Figure 2: High Level Representation of GVT Implementation on FPGA

Figure 2 shows the complete overall design of the I/O Unit. An initial design was conducted and simulated by the PI. For these reasons, we felt comfortable with the aggressive goals set for this one-year project. Unfortunately, we show that we

underestimated the effort required in making the I/O unit operate correctly, which required significant debugging effort, leading to significant redesign relative to the original design.

The physical layer provides the ability to send bit streams over the serial links. In order to exchange meaningful data, a framing standard must be designed and supported. We decided to allow variable length messages in units of 48-bits. While initially this was driven by the PDES application, it makes sense for the most common 32-bit integer data; with 16-bit headers, which can be sent in a single unit.

The I/O unit is designed to work with any application that complies with the packet structure. It is composed of a number of state machines (Figure 3). The first state machine is a 255 entry asynchronous FIFO with a width of 48 bits. It has two purposes. The first is to store multiple packets while the I/O unit is in the act of sending to the all-to-all board. The second purpose is to separate the clock being used by the rest of the FPGA from the I/O clock coming from the all-to-all board. The second state machine is a parallel to serial (P2S) converter. It takes a 48-bit packet, and sends it one bit at a time to every other node via the all-to-all board. The third state machine is actually a set of state machines. An array of serial to parallel (S2P) converters accepts the incoming packets from the all-to-all board. When a packet has been completely received, it is sent to the next state machine, an array of FIFOs. Like the first FIFO in the I/O unit, these FIFOs serve the dual purpose of storage and synchronization. A round robin scheduler selects a message from the FIFOs upon request from outside of the I/O unit.
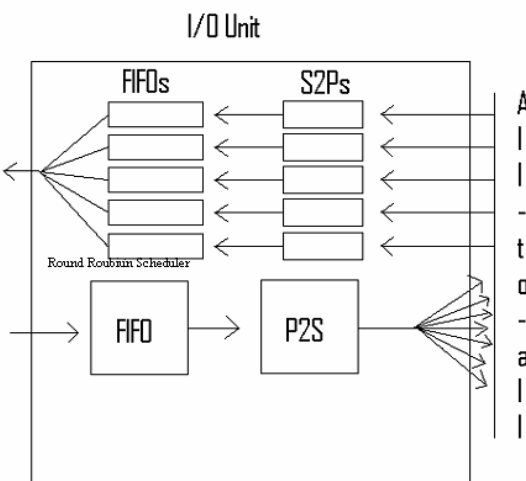

Figure 3—FIFOs and P2S/S2P converters

A message is passed through the I/O unit as follows:
1. The packet is sent from another location on the FPGA card to the I/O unit, where it is stored in the first FIFO.
2. The P2S converter takes the packet when it is empty and sends it one bit at a time to all other nodes via the all-to-all board. If there is no packet available in the FIFO, a blank message (consisting of all 1 bits) is sent instead. All 1s are used because if no node is present at a given connection on the all-to-all board, the node receives floating 1s.
3. The packet is received at all other nodes. The I/O unit receives the message using an S2P converter.
4. Upon receipt, the I/O unit checks to make sure the message is not blank (containing all 1s). If not, the message is stored in a FIFO corresponding to the node that it was received from.
5. The round robin scheduler is used to select what message will be output to the FPGA card.

11

In all phases, the problem is complicated by the variable length nature of the message. The length of the message is automatically detected as it is received. The derived length drives counters that process the message chunk-wise to support variable length messages.

There is more to sending a message than just how it travels through the I/O unit. As was mentioned previously, a clock mechanism has to be in place not only to define each bit, but also to synchronize each of the nodes with one another. In order to do this, every node sends its FPGA clock pulse to the clock node. When the clock node detects the presence of a clock pulse from a node, it registers it to begin receiving its own "I/O clock" pulse. For the sake of synchronization, this clock pulse is not first sent until the internal counter of the I/O clock node resets, which occurs every 48 clock ticks. Because each node keeps its own mod-48 counter and performs its I/O functions based on that counter, each node is synchronized with the others.

The design supports a single message format of variable size that can be thought of as the link layer frame for the design. Each message is made up of units of 48 bits units (packets), as well as a single bit used for framing (if 1, this is the first packet of a message, if 0, it is not). The message description in VHDL is as follows for the first packet of a message:

alias frame_bit : std_logic is s2p_reg(48);

alias src   : std_logic_vector is s2p_reg(47 downto 43);
alias dest   : std_logic_vector is s2p_reg(42 downto 36);
alias length   : std_logic_vector is s2p_reg(35 downto 32);
alias data_32 : std_logic_vector is s2p_reg(31 downto 0);

Note that we include the src, even though it is derivable from the port of the incoming message; the thought was to keep the src for upper layers to help in handling the message. However, this is a field that can be optimized away to free up some header bits for additional purposes (for example, to increase the maximum length of a message). Also, please note that the destination field is larger than the src field to allow for special messages in the future (e.g., broadcast, or grouping). Please check the source for the latest headers.

The rest of the packets making up the message, if any, consists only of data:

alias data_48   : std_logic_vector is s2p_reg (47 downto 0);

Thus, the design provides this basic message type, allowing users to send/receive variable size messages. The API is defined at the level of the I/O unit (internally) rather than at the level of the host processor since we envision most communication to be needed directly between the FPGAs (otherwise the I/O overheads come into play). This also frees up the application developers to build their own message types and send and receive semantics on top of it. For example, in our application, the host processor provides a message count and LVT time update. For our application, we require that this be broadcast to all nodes; so the message is replicated to all outgoing FIFOs

concurrently (correct headers are constructed concurrently). Similarly, the I/O unit API allows sends to a single host, which we did not exercise in our design. On the receive side, currently an outside state machine pulls data from the I/O unit to check for updates. The I/O unit signals the presence of data (in any of the FIFOs). Since for this application we do not care who the update comes in –all need to be processed—we simply pull data when there is room for it for internal consumption in the GVT component. A version of the design (for example using generics) that requests a receive from a specific destination can be built. In addition, the current round robin scheduler can be moved outside the I/O unit, which then exposes a bit vector of which FIFOs have messages, decoupling the receive semantics from the I/O unit implementation. Also, the pulled messages can be sent to the host processor by setting an interrupt (which we do for the final result of the GVT computation). Finally, additional functionality like ARQ, flow control, etc… can be built by higher layers if desired using standard layering approaches.

The FIFO is currently defined with that static width. The parallel to serial and serial to parallel state machines are size customizable, and improvements to the FIFO could change the current state. However, headers will be static, and application designers will have to keep this in mind. A set of related packets is called a message. Each message can be made up of up to sixteen packets in the current design. Each message has a 16-bit header contained in the first packet. Every other packet in the message contains only data. The header must include the number of packets in the message, but from there, can include application specific information like the type of message being sent, what nodes are to receive the message, and similar data.

The original plan for developing the FIFO was fairly straightforward. The original code supporting this aspect of operation had been tested using the Modelsim simulator and was shown to be working. Unfortunately, when we attempted to integrate this implementation with the new LVDS and all-to-all board code, very little synthesized correctly. Through testing, we determined that some of the state machines worked correctly within the design while others did not due to the asynchronous clocking from multiple unsynchronized clock sources. This difficult pitfall of logic design created unpredictable race conditions and was not accounted for in the original design. Further, it is very hard to account for these effects in simulation; for example, the original design simulated the I/O card using a clock synchronized with the internal clock and did not catch any of the race conditions for this reason. We were able to keep the basic layout for the I/O unit, and a number of the state machines, but had to debug and rewrite significant amounts of code in other places. We had not planned for such an extensive revision of the initial code, and this aspect of the effort put us significantly behind schedule.

After completing the code for each state machine, it was synthesized and loaded onto the FPGA card to test it to ensure that it worked. While each of the state machines worked individually, when everything was brought together, they did not integrate smoothly. After spending much time in debugging these problems, it became apparent that race conditions across state machines also existed. Essentially, while no sections of code were using two clocks at the same time, some state machines using two different clocks were more coupled than they should have been, resulting in some of the

errors. This led to the third and current design.

Unfortunately, the design is not fully correctly operational yet. Below we discuss the remaining problems. Despite spending very large amounts of time, well beyond the lifetime of the project, we were not able to completely debug them. A main limitation was the unusability of the x-based tools hosted on the AFRL machines remotely. As a result, we had to rely on trial and error with synthesized code, which is not cleanly observable or controllable. In the end, we ran out of time.

First, the round robin scheduler has not been tested after synthesis. This is because, without being able to successfully send packets, we would have been unable to use the scheduler to check how many packets were in any given message. The round robin scheduler was tested successfully in simulation, but is currently a bottleneck in that it cannot be clocked faster than 50MHz. However, it can be easily replaced by an off the shelf priority decoder design.

There were a few other problems in the code. First, somewhere in the I/O unit, the message being read from the host was being dropped. All of the individual state machines worked when tested alone, so we have not figured out why this was happening. Through testing though, it appears that the message is dropped in the P2S converter, but I could not determine a reason or a solution. It did appear that the synthesizer being used for the VHDL code was optimizing the message away, but for no reason that could be determined. It was also difficult to figure out if this was actually the case. Because I was having problems running applications using the x-server remotely, I was not able to test this in the VHDL simulator. For the sake of testing, we generated a message to send over the board from within the P2S state machine.

Another unsolved problem is the clock synchronization. While the code is in place, the nodes do not seem to be synchronizing with one another. The result is that while messages are being sent and received, the nodes are receiving the messages with the bits shifted. That is to say, if the following binary byte was sent: "10011000", it could have been received as "00100110", or any other shift. The shifting was based on at what given times the nodes were started and were different for every run. Again, I was unable to determine the cause or a solution. All other parts of the I/O unit have successfully been tested. This includes the template to interact with the host.

One final unsolved problem had to do with the number of nodes that we could use. When working with all 24 nodes, the percentage of the FPGA card being used at any one node was high. This was caused by the FIFOs that stored the incoming data, of which there were 23, each with 255 entries with a width of 48 bits. As a result, the FPGA cards were being overused and outputting messages incorrectly. More specifically, there is a limited number of memory elements on the FPGA suitable for use as FIFOs; the large size of the FIFOs combined with the number of ports led to high utilization of these distributed resources, causing the design the span the whole FPGA chip and increasing wiring delays. We note that this is just a conjecture, as the simulation models did not exhibit this problem, which is only observed in the synthesized model. For testing purposes, we reduced the number of nodes being used to 8. The solution was successful in the short run, but for long-term purposes, the

FIFOs would need to be replaced with smaller versions. Having 255 entries per FIFO is certainly not necessary, and that number could easily be reduced to 50 without fear of overflow. However, despite spending a significant amount of time working with the problem, I was unable to either find or design a FIFO that was suitable for the task.

7   SPEEDES and GVT Calculation

The initial plan called for using the developed all-to-all communication support in accelerating PDES. From the perspective of the FPGA, the design of the GVT calculator is fairly simple and straightforward. Twenty-three arrays are kept in the calculator, each representing one of the remote nodes. The newest LVT [local virtual time] and the difference between the number of sent and received messages is kept here. As LVT is updated or message counts change, the host updates its local values. When the total number of messages in transit reaches 0, GVT is computed by taking the minimum LVT. The building blocks for the GVT have been tested in simulation and work correctly.

The state machines that connect the I/O unit to the GVT calculator were not completely tested after synthesis because finishing the I/O unit was a higher priority, but these machines should be fairly simple in nature. The I/O unit will notify the GVT calculator when it has a message. The GVT calculator will request that message, which will include the node that it was received from. The headers will be stripped away and the LVT will be stored within the calculator. Another state machine is responsible for requesting the calculated GVT and returning it to the local host. It notifies the GVT calculator that a request has been received from the local host and requests the GVT. When the calculator is ready to return the result, it notifies the state machine, which takes the result and returns it to the host.

Upon completion of the GVT calculator-supporting state machines, the design would have to be incorporated into the SPEEDES simulator, which is a difficult, though achievable objective. The first step is to locate all places where SPEEDES uses breathing time warp and requests the LVT of other nodes so that it can calculate the GVT. Because of the size of the SPEEDES code, this is a difficult challenge. Once this has been done, the code used to request that the FPGA return a GVT calculation can replace the old code. When a request has been made, simulation can continue, but only locally. Until a new GVT has been returned, we do not want to commit any of the actions in the simulator. We could not be more specific than that until the GVT calculator itself was more complete.

8   Conclusion

Unfortunately, we were not able to finish the project as planned. This was the result of a number of unexpected problems that continually put us further behind schedule. The first of these problems, as described earlier, was the difficulty in locating software that would enable me to work with the LVDS cards so that we could send messages over the all-to-all board, and the associated learning curve in using the template, once provided. In addition, there were problems with the FPGA cards at a few nodes, which

meant that we had to change the connections between the board and the FPGA card a number of times. This led to a number of tests being interrupted mid-run, a problem because of the length of the tests being run. However, we were eventually successful in obtaining test results.

The next major delay occurred when we realized that we would not be able to use nearly as much of the successfully simulated I/O unit code in the synthesized design for the FPGA card. This led to the first rewrite of the code that kept the basic design for the I/O unit but replaced a number of the state machines. The three-clock synchronization problem that we encountered led to a second rewrite of the design. While neither rewrite included a significant amount of coding, both decisions were arrived at through significant amounts of testing and involved a great deal more testing during the rewrite to ensure that each piece was working correctly.

The final set of problems encountered was on the second rewrite and is also mentioned above. These problems include clock synchronization, expansion to 24 nodes, and the dropping of packets within the I/O unit. While we are confident that the design for the I/O board could successfully work, we were unable to solve these problems within the given amount of time and without having access to tools. A combination of factors led to this. First, we had fallen significantly behind schedule because of earlier problems. Second, we had to work remotely, which slowed down development significantly. The HHPC x-server runs slowly remotely to the point where they are not usable interactively (the simulator). This necessitated trial-and-error debugging by synthesizing code and seeing what happens. This is extremely inefficient as the code synthesis is very slow and the hardware is difficult to observe and control.

Many of the problems we faced were post-synthesis, making debugging extremely difficult. In addition, absent access to debugging tools (which we did not have, due to the un-usable remote X-interface), it became almost impossible to debug the design. We believe that the whole design cycle would have been accelerated with access to a modern tool-chain. We believe that with access to such tools, and more time, the design can be made to work—we have specific examples that isolate the problems that can be used to debug them that can serve as a starting point to address these shortcomings.

Getting the PDES study "working" after the I/O unit design is operation is fairly straightforward: it requires:
1. Replacing the logic for GVT computation, which is based on a fuzzy barrier, with a check of the status of the on-going GVT computation.
2. Sending an update of LVT and message counts any time a message is received
3. Converting Time objects from the simulator to 64 bit representation used on the FPGA. This entails loss in resolution, but 64 bits are sufficient for most applications.

However, in order to get the implementation working efficiently requires structural changes to the simulator to allow pipelined computation of GVT, to allow event processing to continue while GVT is being computed. This requires adding a color flag to all messages and keeping track of message counts for each color, LVT for each

color and manage color transitions triggered by GVT computation. We have the algorithm developed with a partial implementation (approximately 75% of the implementation – integrating with the simulation loop remains).

9. References

[Avril96]   H. Avril and C. Tropper, "The dynamic load balancing of clustered time warp for logic simulation," Workshop on Parallel and Distributed Simulation (PADS), 1996, Pages 20-27.

[Ball 1990] D. Ball and S. Hoyt, "The Adaptive Time-Warp Concurrency Control Algorithm", Proceedings of the SCS Multi-conference on Distributed Simulation 22(1), 174-177

[Burdorf 1993] C. Burdorf and J. Marti, "Load Balancing Strategies for Time Warp on Multi-User Workstations," The Computer Journal 36(2), 168-176

[Carothers 1996] C. Carothers and R. Fujimoto, "Background Execution of Time Warp Programs," Proc. Of the 10th Workshop on Parallel and Distributed Simulation (PADS 1996), 12-19

[Das 1997] S. Das and R. Fujimoto, "Adaptive Memory Management and Optimism Control in Time Warp," ACM Transactions on Modeling and Computer Simulation (TOMACS), 7(2), 239-271

[Ferscha 1995] A. Ferscha, "Probabilistic Adaptive Direct Optimism Control in Time Warp", Proc. Of PADS 1995, pages 120-129

[Fujimoto 1990]   R. Fujimoto, "Parallel Discrete Event Simulation", Communications of the ACM, 33(10):30-53, Oct. 1990.

[Glazer 1993] D. Glazer and C. Tropper, "On Process Migration and Load Balancing in Time Warp," IEEE Transactions on Parallel and Distributed Systems, 1993

[Metron] Metron, Inc., "The SPEEDES API Reference Manual", 2002 (available from http://www.speedes.com)

[Jefferson 1983]   D. Jefferson, "Virtual Time", Proc. Of the International Conference on Parallel Processing, 1983, pages 384-394

[Jones 1998] K. Jones and S. Das, "Combining Optimism Limiting Schemes in Time Warp Based Parallel Simulation," Proc. Winter Simulation Conference 1998.

[Chetlur 1998] M. Chetlur, N. B. Abu-Ghazaleh, R. Radhakrishnan, P. A. Wilsey: Optimizing Communication in Time-Warp Simulators. Workshop on Parallel and Distributed Simulation 1998: 64-71

[Reiher 1990] P. Reiher and D. Jefferson, "Virtual time Based Dynamic Load Management in the Time Warp Operating System," Proc. Of the SCS Multiconference on Distributed Simulation 22(1), 103-111

[Sokol 1990] L. Sokol and B. Stucky, "MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm," Proceedings of the SCS Multi-conference on Distributed Simulation 22(1), 169-173

[Sharma 1999] Girindra D. Sharma, Radharamanan Radhakrishnan, Umesh Kumar V. Rajasekaran, Nael B. Abu-Ghazaleh, Philip A. Wilsey: Time Warp Simulation on Clumps. Workshop on Parallel and Distributed Simulation 1999: 174-181

[Turner 1992] S. Turner and M. Xu, "Performance Evaluation of the Bounded Time-Warp Algorithm," Proc. Of the SCS Multiconference on Parallel and Distributed Simulation 24(3), 117-126

[Abu-Ghazaleh 2004] N. Abu-Ghazaleh, R. Linderman, R. Hillman and J.Hanna, "Using a Heterogeneous High Performance Cluster for Parallel Discrete Event Simulation", High Performance Computing User Group, 2004.

[Steinman 1993] J.S. Steinman. "Breathing Time Warp". In 7th Workshop on Parallel and Distributed Simulation (PADS'93), pages 109--118, May 1993

[Radhakrishnan 1998] Radharamanan Radhakrishnan, Nael B. Abu-Ghazaleh, Malolan Chetlur, Philip A. Wilsey: On-line Configuration of a Time Warp Parallel Discrete Event Simulator. International Conference on Parallel Processing. ICPP 1998: 28-38

[Boden 1995] Boden, D. Cohen, R. Felderman, A.Kulawik, C. Seitz, J.Seizovic and W-K. Su, "Myrinet: A Gigabit-per-second Local Area Network Network", IEEE Micro, 15(1):29--36, February 1995.